H-0261 (H0808-003) August 11, 2008
Other

IBM Research Report

Proposed New Appendix B for IEEE 1850 (PSL)

Cindy Eisner, Dana Fisman*
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

*Also with Hebrew University

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Proposed New Appendix B for IEEE 1850 (PSL)

Cindy Eisner[1] and Dana Fisman[1,2]

[1] IBM Haifa Research Lab
[2] Hebrew University

August 7, 2008

2008-08-07 22:31

Abstract

This proposal for a new Appendix B is based on the document "Simplified Proposal for Extending the Formal Semantics of PSL with Local Variables, Procedural Blocks, Past Expressions and Clock Alignment Operators and Including a Solution to Structural Contradictions" as sent to the subcommittee on February 14, 2008.

Annex B

(normative)

Formal Syntax and Semantics of IEEE Std 1850 Property Specification Language (PSL)

This appendix formally describes the syntax and semantics of the temporal layer.

B.1 Typed-text representation of symbols

Table 1 shows the mapping of various symbols used in this definition to the corresponding typed-text PSL representation, in the different flavors.

NOTE –
For reasons of simplicity, the syntax given herein is more flexible than the one defined by the extended BNF (given in Annex A). That is, some of the expressions which are legal here are not legal under the BNF Grammar. Users should use the stricter syntax, as defined by the BNF grammar in Annex A.

B.2 Syntax

The logic PSL is defined with respect to a non-empty finite set of atomic propositions $\mathcal{P}$, a finite set of local variables $\mathcal{V}$ with a finite domain $\mathcal{D}$, a given set of (unary and binary) operators $\mathcal{O}$ over the domain $\mathcal{D}$. We assume that T and F belong to $\mathcal{D}$.

The definition of Boolean expressions, expressions and SEREs is given by mutual induction.

Definition 1 (Boolean Expressions) *Let $\odot_{\mathrm{B}} \in \mathcal{O}$ and $\otimes_{\mathrm{B}} \in \mathcal{O}$ be unary and binary operators such that $\odot_{\mathrm{B}} : \mathcal{D} \mapsto \{\mathrm{T}, \mathrm{F}\}$ and $\otimes_{\mathrm{B}} : \mathcal{D}^2 \mapsto \{\mathrm{T}, \mathrm{F}\}$. Let $p \in \mathcal{P}$ and let k be a non-negative integer. Let e be an expression and r a SERE.*

Table 1: Typed-text symbols in the SystemVerilog, Verilog, VHDL, SystemC and GDL flavors

	SystemVerilog	Verilog	VHDL	SystemC	GDL
$\mapsto$	`|->`	`|->`	`|->`	`|->`	`|->`
$\Mapsto$	`|=>`	`|=>`	`|=>`	`|=>`	`|=>`
$\rightarrow$	`->`	`->`	`->`	`->`	`->`
$\leftrightarrow$	`<->`	`<->`	`<->`	`<->`	`<->`
$\neg$	`!`	`!`	`not`	`!`	`!`
$\wedge$	`&&`	`&&`	`and`	`&&`	`&`
$\vee$	`||`	`||`	`or`	`||`	`|`
..	`:`	`:`	`to`	`:`	`..`
$\langle\ \rangle$	`[ ]`	`[ ]`	`( )`	`( )`	`( )`
$\leftarrow$	`<=`	`<=`	`<=`	`=`	`:=`
$\cup$	|	|	|	|	|
$\cap$	&&	&&	&&	&&	&&
$\cdot$	;	;	;	;	;
$\circ$	:	:	:	:	:
Z$\leftarrow$E	`[[` equivalent sequence of assignments in the flavor language `]]`				
var(Z)	`[[` declaration of Z in the flavor language `]]`				
free(Z)	`[[` free(Z) `]]`				

The set of Boolean expressions $\mathcal{B}$ *is defined inductively as follows:*

$$b ::= \quad p \quad | \quad \odot_{\mathrm{B}}\, e \quad | \quad e \otimes_{\mathrm{B}} e \quad | \quad prev(b,k) \quad | \quad ended(r)$$

We refer to a Boolean expression that does not use local variables and does not refer to the past (that is, does not use prev or ended) as a basic Boolean expression*.*

We use true and false as abbreviations for $p \vee \neg p$ and $p \wedge \neg p$, respectively, where p is some atomic proposition, and $\vee, \wedge \in \otimes_{\mathrm{B}}$ and $\neg \in \odot_{\mathrm{B}}$ are defined in the usual way.

Definition 2 (Expressions) *Let $b \in \mathcal{B}$ be a Boolean expression and $y \in \mathcal{V}$ be a local variable. Let $\odot \in \mathcal{O}$ and $\otimes \in \mathcal{O}$ be unary and binary operators, respectively, such that $\odot : \mathcal{D} \mapsto \mathcal{D}$ and $\otimes : \mathcal{D}^2 \mapsto \mathcal{D}$. Let k be a non-negative integer.*

The set of expressions $\mathcal{E}$ *is defined inductively as follows:*

$$e ::= \quad b \quad | \quad y \quad | \quad \odot\, e \quad | \quad e \otimes e \quad | \quad prev(e,k)$$

Definition 3 (Sequential Extended Regular Expressions (SEREs)) *Let e be an expression and d a value in $\mathcal{D}$. Let* Z *be a (possibly empty) finite sequence of local variables. Let* E *be a sequence of expressions of the same length as* Z*. Let b be a Boolean expression.The set of* SEREs *is defined inductively as follows:*

$$\begin{array}{rcl} r ::= & [*0] \ | \ e{=}d, \mathrm{Z}{\leftarrow}\mathrm{E} \ | \ r \cdot r \ | \ r \circ r \ | \ r[+] \ | \ r \ \cup \ r \ | \ r \ \cap \ r \\ & \{r\} \ | \ \{var(\mathrm{Z})\ r\} \ | \ \{free(\mathrm{Z})\ r\} \ | \ r@b \ | \end{array}$$

We sometimes use __ to denote the empty assignment, i.e. an assignment of the form Z$\leftarrow$E where Z is an empty sequence. For an expression e and value d, we use $e{=}d$ to abbreviate $e{=}d, \_\_$ and we use e to abbreviate $e{=}\mathrm{T}$ and $e{=}\mathrm{T}, \_\_$.

Definition 4 (Formulas) *Let k be a non-negative integer, r a SERE, $\mathrm{Z} \subseteq \mathcal{V}$ a finite set of local variables. Let b be a Boolean expression. Let t be a SERE that does not refer to local variables. The set of FL formulas is defined inductively as follows:*

$$\begin{array}{rcl} \varphi ::= & r! \ | \ r \ | \ r \mapsto \varphi \ | \ (\varphi) \ | \ \neg\varphi \ | \ \varphi \wedge \varphi \\ & X![k]\ \varphi \ | \ \varphi\ U\ \varphi \ | \ \varphi\ sync\_abort\ t \ | \ (var(\mathrm{Z})\ \varphi) \ | \ \varphi@b \end{array}$$

NOTE –
The formula φ sync_abort t is only accessible to the user when t is a Boolean expression.

Definition 5 (Formulas of the Optional Branching Extension (OBE)) *Let b be a basic Boolean expression. The set of OBE formulas is defined inductively as follows:*

$$f ::= \quad b \mid (f) \mid \neg f \mid f \wedge f \mid \mathsf{EX}f \mid \mathsf{E}[f\ \mathsf{U}\ f] \mid \mathsf{EG}f$$

Definition 6 (PSL Formulas)

1. *Every FL formula is a PSL formula.*
2. *Every OBE formula is a PSL formula.*

In Section B.4, we show additional operators which provide syntactic sugaring to the ones above.

B.3 Semantics

The semantics of PSL formulas are defined with respect to a *model*. A model is a quintuple $(S, S_0, R, \mathcal{P}, L)$, where S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $R \subseteq S \times S$ is the transition relation, $\mathcal{P}$ is a non-empty set of atomic propositions, and L is the valuation, a function $L : S \longrightarrow 2^{\mathcal{P}}$, mapping each state with a set of atomic propositions valid in that state.

A *path* π is a possibly empty finite (or infinite) sequence of states $\pi = (\pi_0, \pi_1, \pi_2, \cdots, \pi_n)$ (or $\pi = (\pi_0, \pi_1, \pi_2, \cdots)$). A *computation path* π of a model M is a non-empty finite (or infinite) path π such that $\pi_0 \in S_0$ and for every $i < n$, $R(\pi_i, \pi_{i+1})$ and for no s, $R(\pi_n, s)$ (or such that for every i, $R(\pi_i, \pi_{i+1})$). Given a finite (or infinite) path π, we overload L, to denote the extension of the valuation function L from states to paths as follows: $L(\pi) = L(\pi_0)L(\pi_1)\ldots L(\pi_n)$ (or $L(\pi) = L(\pi_0)L(\pi_1)\ldots$). Thus, we have a mapping from *states* in M to *letters* of $2^{\mathcal{P}}$, and from finite (or infinite) *paths* in M to finite (or infinite) *words* over $2^{\mathcal{P}}$.

B.3.1 Semantics of FL formulas

We first define the semantics without the clock operator (@), local variables, or expressions that refer to the past (*prev*() and *ended*()). Then we define the full semantics.

B.3.1.1 The Basic Semantics

The basic semantics are the semantics of formulas without the clock operator (@), local variables or expressions that refer to the past (*prev*() and *ended*()). In the basic semantics, $\mathcal{D} = \{\mathsf{true}, \mathsf{false}\}$ and all expressions are basic Boolean expressions.

We denote an element of $\mathcal{P}$ by p and an element of $\mathcal{D}$ by d. Let Σ be the set of all possible assignments to the atomic propositions $\mathcal{P}$ (i.e., $\Sigma = 2^{\mathcal{P}}$). We denote an element of Σ by σ and we use $\sigma(p)$ to denote the truth value given to p by σ.

We use u, v, w to denote (possibly empty) words over Σ. We use a, b, c to denote letters over Σ. We use ϵ to denote the empty word. We use $\cdot$ and $\circ$ to denote concatenation and concatenation with one overlapping letter, respectively. Formally, given languages L_1 and L_2 we use $L_1 \cdot L_2$ to denote the set $\{w_1 w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}$. We use $L_1 \circ L_2$ to denote the set $\{w_1 \ell w_2 \mid w_1 \ell \in L_1 \text{ and } \ell w_2 \in L_2\}$. For a language L we use L^0 to denote $\{\epsilon\}$. We use L^i to denote $L^{i-1} \cdot L$. We use L^* and L^+ to denote $\bigcup_{i \geq 0} L^i$ and $\bigcup_{i > 0} L^i$, respectively. We use L^ω for the language composed of infinitely many concatenations of L with itself. We use L^∞ for the union $L^* \cup L^\omega$.

For a finite/infinite word w where $w = a_0 a_1 \cdots a_n$ or $w = a_0 a_1 a_2 \cdots$ and integers i, j we use w^i to denote the $(i+1)^{th}$ letter of w. That is, $w^i = a^i$. We use $w^{i\cdot\cdot}$ to denote the suffix of w starting at w^i. That is, $w^{i\cdot\cdot} = a_i a_{i+1} \cdots a_n$ or $w^{i\cdot\cdot} = a_i a_{i+1} \cdots$. We use $w^{i\cdot\cdot j}$ to denote the finite word starting

at position i and ending at position j. That is, $w^{i..j} = a_i a_{i+1} \cdots a_j$. We make use of an "overflow" and "underflow" for the positions of w. That is, $w^{i..}$ and $w^{i..j}$ are defined for i bigger than the last position of w and for $j < i$ as follows: $w^{i..} = w^{i..j} = \epsilon$.

For $d \in \{\text{T}, \text{F}\}$ and a (Boolean) expression e, the semantics of $e = d$, denoted $\mathcal{B}(e = d)$, is the language of all letters a on which $e = d$ holds. It is defined formally as follows.

Definition 7 (The Basic Boolean Semantics)

- *Base case:*
  - $\mathcal{B}(p{=}d) = \{a \mid a(p) = d\}$
- *Standard operators:*
  - $\mathcal{B}(\odot e{=}d) = \{a \mid \exists d' {\in} \mathcal{D}$ *s.t.* $\odot(d'){=}d$ *and* $a {\in} \mathcal{B}(e{=}d')\}$
  - $\mathcal{B}(e_1 \otimes e_2{=}d) = \{a \mid \exists d_1, d_2 \in \mathcal{D}$ *s.t.* $(d_1 \otimes d_2){=}d$ *and* $a {\in} \mathcal{B}(e_1{=}d_1)$ *and* $a {\in} \mathcal{B}(e_2{=}d_2)\}$

For a regular expression r the basic *language* of r, denoted $\mathcal{L}(r)$, is defined as follows, where $\_\_$ denotes an empty sequence of assignments.

Definition 8 (The Basic Language of SEREs)

- *Base cases:*
  - $\mathcal{L}([*0]) = \epsilon$
  - $\mathcal{L}(e{=}d, \_\_) = \mathcal{B}(e{=}d)$
- *Standard SERE operators:*
  - $\mathcal{L}(\{r\}) = \mathcal{L}(r)$
  - $\mathcal{L}(r_1{\cdot}r_2) = \mathcal{L}(r_1){\cdot}\mathcal{L}(r_2)$
  - $\mathcal{L}(r_1{\circ}r_2) = \mathcal{L}(r_1){\circ}\mathcal{L}(r_2)$
  - $\mathcal{L}(r[+]) = \mathcal{L}(r)^+$
  - $\mathcal{L}(r_1 \ \cup \ r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$
  - $\mathcal{L}(r_1 \ \cap \ r_2) = \mathcal{L}(r_1) \cap \mathcal{L}(r_2)$

The basic *language of proper prefixes* and the basic *loop language* of r, denoted $\mathcal{F}(r)$ and $\mathcal{I}(r)$ respectively, will be used to define the semantics of a formula r (a weak SERE formula). Intuitively, $\mathcal{F}(r)$ consists of *finite* proper prefixes of words in $\mathcal{L}(r)$, except that logical contradictions such as false and structural contradictions such as $\{b\} \cap \{b{\cdot}b\}$ are considered satisfiable, and $\mathcal{I}(r)$ consists of *infinite* words in which we get "stuck forever" in a starred sub-expression of r.

Definition 9 (The Basic Language of Proper Prefixes of SEREs)

- *Base cases:*
  - $\mathcal{F}([*0]) = \emptyset$
  - $\mathcal{F}(e = d, \_\_) = \epsilon$
- *Standard SERE operators:*
  - $\mathcal{F}(\{r\}) = \mathcal{F}(r)$
  - $\mathcal{F}(r_1{\cdot}r_2) = \mathcal{F}(r_1) \cup (\mathcal{L}(r_1){\cdot}\mathcal{F}(r_2))$
  - $\mathcal{F}(r_1{\circ}r_2) = \mathcal{F}(r_1) \cup (\mathcal{L}(r_1){\circ}\mathcal{F}(r_2))$
  - $\mathcal{F}(r[+]) = \mathcal{L}(r)^*{\cdot}\mathcal{F}(r)$
  - $\mathcal{F}(r_1 \ \cup \ r_2) = \mathcal{F}(r_1) \cup \mathcal{F}(r_2)$
  - $\mathcal{F}(r_1 \ \cap \ r_2) = \mathcal{F}(r_1) \cap \mathcal{F}(r_2)$

Definition 10 (The Basic Loop Language of SEREs)

- *Base cases:*
  - $\mathcal{I}([*0]) = \emptyset$
  - $\mathcal{I}(e{=}d, \_\_) = \emptyset$
- *Standard SERE operators:*
  - $\mathcal{I}(\{r\}) = \mathcal{I}(r)$
  - $\mathcal{I}(r_1{\cdot}r_2) = \mathcal{I}(r_1) \cup (\mathcal{L}(r_1){\cdot}\mathcal{I}(r_2))$
  - $\mathcal{I}(r_1{\circ}r_2) = \mathcal{I}(r_1) \cup (\mathcal{L}(r_1){\circ}\mathcal{I}(r_2))$
  - $\mathcal{I}(r[+]) = (\mathcal{L}(r)^*{\cdot}\mathcal{I}(r)) \cup \mathcal{L}(r)^\omega$
  - $\mathcal{I}(r_1 \ \cup \ r_2) = \mathcal{I}(r_1) \cup \mathcal{I}(r_2)$
  - $\mathcal{I}(r_1 \ \cap \ r_2) = \mathcal{I}(r_1) \cap \mathcal{I}(r_2)$

We define the semantics of a formula with respect to a word w and a context indicating the view $\upsilon \in \{\text{S}, \text{N}, \text{W}\}$ (where S, N, W correspond to the *strong*, *neutral* and *weak* views, respectively). For a view υ we use $\overline{\upsilon}$ to denote the *dual view* which is S if $\upsilon = \text{W}$, W if $\upsilon = \text{S}$, and υ otherwise (if $\upsilon = \text{N}$). We use j and k below to denote non-negative integers.

The base cases for the definitions of a formula are strong and weak SEREs: $r!$ and r. A strong SERE $r!$ holds under the neutral or strong view if it has a non-empty prefix in the language of r. A strong SERE $r!$ holds under the weak view if it holds under the neutral view or the word "'ended too soon" — that is, the word is a proper prefix of a word in the language of r (formally, $w \in \mathcal{F}(r) \cup \{\epsilon\}$). A weak SERE r holds on a given word if the strong SERE r holds on that word or the word "got stuck forever" in a starred sub-SERE of r. That is, $w \in \mathcal{I}(r)$. In addition, under the weak and neutral views the word may "end too soon".

The relation $w \models^{\upsilon} \varphi$, read "$w$ models φ in the view υ under the basic semantics", is formally defined as follows.

Definition 11 (Satisfaction in the Basic Semantics)

I. $w \models^{\upsilon} r! \iff$ *either* $\exists j$ *s.t.* $w^{0..j} \in \mathcal{L}(r)$
or $(\upsilon = \text{W}$ *and* $w \in \mathcal{F}(r) \cup \{\epsilon\})$

II. $w \models^{\upsilon} r \iff$ *either* $\exists j$ *s.t.* $w^{0..j} \in \mathcal{L}(r)$ *or* $w \in \mathcal{I}(r)$
or $(\upsilon \in \{\text{W}, \text{N}\}$ *and* $w \in \mathcal{F}(r) \cup \{\epsilon\})$

III. $w \models^{\upsilon} r \mapsto \varphi \iff \forall j$ *s.t.* $w^{0..j} \in \mathcal{L}(r)$ *we have that* $w^{j..} \models^{\upsilon} \varphi$ *and*
either $\upsilon \in \{\text{W}, \text{N}\}$ *or* $w \notin \mathcal{F}(r) \cup \{\epsilon\}$

IV. $w \models^{\upsilon} (\varphi) \iff w \models^{\upsilon} \varphi$

V. $w \models^{\upsilon} \neg\varphi \iff w \not\models^{\overline{\upsilon}} \varphi$

VI. $w \models^{\upsilon} \varphi \wedge \psi \iff w \models^{\upsilon} \varphi$ *and* $w \models^{\upsilon} \psi$

VII. $w \models^{\upsilon} \mathsf{X!}[k]\,\varphi \iff$ *either* $w^{0..k} \in \mathcal{L}(true\,[k{+}1])$ *and* $w^{k..} \models^{\upsilon} \varphi$
or $(\upsilon = \text{W}$ *and* $w \in \mathcal{F}(true\,[k{+}1]) \cup \{\epsilon\})$

VIII. $w \models^{\upsilon} \varphi\,\mathsf{U}\,\psi \iff \exists k$ *s.t.* $w \models^{\upsilon} \mathsf{X!}[k]\,\psi$ *and* $\forall j < k,\ w \models^{\upsilon} \mathsf{X!}[j]\,\varphi$

IX. $w \models^{\upsilon} \varphi$ *sync_abort* $t \iff$ *either* $w \models^{\upsilon} \varphi$ *or* $\exists j, k$ *s.t.* $w^{j..k} \in \mathcal{L}(t)$ *and* $w^{0..k-1} \models^{\text{W}} \varphi$

Definition 12 (Holds Weakly, Neutrally, Strongly, Pending and Fails in the Basic Semantics) *Let w be a word and φ a PSL formula.*

- φ holds weakly *on w in the basic semantics if* $w \models^{\text{W}} \varphi$.
- φ holds (neutrally) *on w in the basic semantics if* $w \models^{\text{N}} \varphi$.
- φ holds strongly *on w in the basic semantics if* $w \models^{\text{S}} \varphi$.
- φ *is* pending *on word w iff* $w \models^{\text{W}} \varphi$ *and* $w \not\models^{\text{N}} \varphi$
- φ fails *on word w iff* $w \not\models^{\text{W}} \varphi$

NOTES –

I. The semantics given here for the LTL operators under the neutral view and in the absence of non-degenerate SEREs is equivalent to the standard LTL semantics [B10,B11]. In particular, the semantics of X! and U under the neutral view can be equivalently phrased as follows:

- $w \models^{\text{N}} \mathsf{X!}\,[k]\,\varphi \iff |w| > k$ and $w^{k..} \models^{\text{N}} \varphi$
- $w \models^{\text{N}} \varphi\,\mathsf{U}\,\psi \iff \exists k$ s.t. $w^{k..} \models^{\text{N}} \psi$ and $\forall j < k,\ w^{j..} \models^{\text{N}} \varphi$

The semantics of these operators are phrased as they are in Definition 11 since this form is more similar to the one for full semantics.

II. The semantics given here for the LTL operators and the sync_abort operator is equivalent to the truncated semantics given in [B1] where $\models^{\text{S}}$, $\models^{\text{N}}$ and $\models^{\text{W}}$ are denoted $\models^{+}$, $\models$ and $\models^{-}$, respectively, and sync_abort is denoted trunc_w.

III. For FL formulas without the $\cap$ operator, the semantics here is equivalent to the semantics given in the previous version of the standard (IEEE 1850-2005) where words are interpreted over $2^P \cup \{\top, \bot\}$. Then, as shown in [B3], the three following equivalences hold for a formula φ of LTL^{trunc}:

$$w \models^{\mathrm{W}} \varphi \iff w\top^\omega \models \varphi$$
$$w \models^{\mathrm{N}} \varphi \iff w \models \varphi$$
$$w \models^{\mathrm{S}} \varphi \iff w\bot^\omega \models \varphi$$

IV. As in [B1], for an infinite word w and a formula φ the three views coincide. That is, $w \models^{\mathrm{W}} \varphi \iff w \models^{\mathrm{N}} \varphi \iff w \models^{\mathrm{S}} \varphi$ when w is infinite.

V. There is a subtle difference between Boolean negation and formula negation. For instance, consider the formula $\neg b$. If $\neg$ is Boolean negation, then $\neg b$ is equivalent to formula $\{\neg b\}$, which holds on an empty path. If $\neg$ is formula negation, then $\neg b$ is equivalent to formula $\neg\{b\}$, which does not hold on an empty path. Rather than introduce distinct operators for Boolean and formula negation, we instead adopt the convention that negation applied to a Boolean expression is Boolean negation. This does not restrict expressivity, as braces can be used to get formula negation.

VI. For FL formulas including the $\cap$ operator, the semantics here differs from the semantics given in the previous version of the standard (IEEE 1850-2005) for formulas including *structural contradictions* such as $\{a\} \cap \{a{\cdot}a\}$. In the previous version of the standard, the formula $\{a{\cdot}b[*]{\cdot}r\}$ holds weakly on an infinite path such that a holds on the first letter and b on all the rest if r is a logical contradiction such as `false`, but not if r is a structural contradiction such as $\{c\} \cap \{c{\cdot}c\}$. In the semantics presented here, logical and structural contradictions are treated in a consistent manner. We achieve this by eliminating letters $\top$ and $\bot$ and including instead a context which represents the view (strong, neutral or weak) and three languages for a SERE. The language $\mathcal{L}(r)$ corresponds to tight satisfaction of the previous version of the semantics. Intuitively, the *language of proper prefixes* $\mathcal{F}(r)$ consists of finite proper prefixes of words in $\mathcal{L}(r)$ except that logical and structural contradictions are considered satisfiable. The *loop language* $\mathcal{I}(r)$ consists of infinite words in which we get "stuck forever" in a starred subexpression of r. See [B7] for a thorough study of this subject.

B.3.1.2 The Full Semantics

In the full semantics we add the following to the basic semantics: the clock operator (`@`), local variables, and expressions that refer to the past (*prev*() and *ended*()).

B.3.1.2.1 Supporting the clock operator @

In order to support the clock operator `@`, we add a context κ — a Boolean expression that remembers the current clock. In addition, we define three languages *no_tick*, *tick* and *past_tick*, each of which uses the clock context (see Definition 19).

NOTE –
The clocked semantics for the LTL subset follows the clocks paper [B2], with the exception that strength is applied at the SERE level rather than at the propositional level. Furthermore, following [B5] clock alignment operators are added. The clock strong and weak alignment operators are `X![0]` and `X[0]`, respectively.

B.3.1.2.2 Supporting local variables

In order to support local variables we use an enhanced alphabet Υ as defined below, and we add a context Y that remembers the set of local variables currently in scope.

We denote an element of $\mathcal{P}$ by p, an element of $\mathcal{V}$ by x, y or z, and an element of $\mathcal{D}$ by d. Recall that $\mathcal{D}$ can be any finite superset of $\{\mathrm{T}, \mathrm{F}\}$. Let Σ be the set of all possible assignments to the atomic propositions $\mathcal{P}$ (i.e., $\Sigma = 2^{\mathcal{P}}$). Let Γ be the set of all possible valuations of the local variables in $\mathcal{V}$ (i.e., $\Gamma = \mathcal{D}^{\mathcal{V}}$). We denote an element of Σ by σ and an element of Γ by γ. We use $\sigma(p)$ to denote the truth value given to p by σ, and $\gamma(y)$ to denote the value given to y by γ.

We use Υ to denote the alphabet $\Sigma \times \Gamma \times \Gamma$. We use $\mathtt{u}, \mathtt{v}, \mathtt{w}, \mathtt{h}$ to denote (possibly empty) words over Υ. We use $\mathtt{a}, \mathtt{b}, \mathtt{c}$ to denote letters over Υ. We refer to letters and words over Υ as *enhanced letters* and *enhanced words*, respectively. Let $\mathtt{w} = (\sigma_0, \gamma_0, \gamma_0')(\sigma_1, \gamma_1, \gamma_1') \cdots$ be an enhanced word.

We use $\mathtt{w}|_\sigma$,$\mathtt{w}|_\gamma$,$\mathtt{w}|_{\gamma'}$ to denote the word obtained from $\mathtt{w}$ by leaving only the first, second or third component, respectively. That is $\mathtt{w}|_\sigma = \sigma_0\sigma_1\cdots$, $\mathtt{w}|_\gamma = \gamma_0\gamma_1\cdots$ and $\mathtt{w}|_{\gamma'} = \gamma'_0\gamma'_1\cdots$.

The intuitive roles of the components of a letter are as follows. The first component of a letter $\mathtt{a}$ (i.e. $\mathtt{a}|_\sigma$) holds the valuation of the atomic propositions at the given cycle. The second component of a letter $\mathtt{a}$ (i.e. $\mathtt{a}|_\gamma$) holds the current values of the local variables at the given cycle (the *pre-value*). The third component of a letter $\mathtt{a}$ (i.e. $\mathtt{a}|_{\gamma'}$) holds the values of the local variables after the assignments have taken place (the *post-value*).

NOTE –
The semantics for local variables follows the semantics proposed in [B6,B8].

B.3.1.2.3 Supporting past expressions *prev*() and *ended*()

In order to support past expressions *prev*() and *ended*() we use *words with history* defined as follows.

We refer to a pair of words $(\mathtt{h},\mathtt{w})$ as a word with history (or *h-word* for short) where intuitively $\mathtt{h}$ is the history and $\mathtt{w}$ is the present and future, with the first letter of $\mathtt{w}$ being the present. We refer to a pair $(\mathtt{h},\mathtt{a})$ as an *h-letter*. We use $\mathfrak{w}$ and $\mathfrak{B}$ to denote h-words.

For standard words w_1 and w_2 from some alphabet, concatenation means simply placing the letters of w_2 after the letters of w_1 to get the word $w = w_1w_2$. Concatenating h-words is similar, but we must account for history. Intuitively, the historical component $\mathtt{h}$ of an h-word $(\mathtt{h},\mathtt{w})$ records the historical context of $\mathtt{w}$, for use in case we encounter a *prev* or *ended* operator. For concatenation to be consistent, then, concatenating h-words $(\mathtt{h}_1,\mathtt{w}_1)$ and $(\mathtt{h}_2,\mathtt{w}_2)$ should only be possible if $\mathtt{h}_2$ correctly records the history seen by the resulting word just before it encounters $\mathtt{w}_2$. That is, if $\mathtt{h}_2 = \mathtt{h}_1\mathtt{w}_1$. Fusion of h-words is similar to concatenation, except that there is a one-letter overlap between the words being concatenated, where intuitively, the overlap letter is actually the merging of two separate letters such that the σ components of the two letters agree, but the γ and γ' components do not necessarily agree. This allows us to support an assignment taking place "in the middle of a letter", as we desire.

Definition 13 (Concatenation and fusion of languages of h-words) *Let L_1 and L_2 be sets of h-words. Their concatenation and fusion is defined as follows, where $\cdot$ and $\circ$ denote concatenation and fusion, respectively.*
– $L_1{\cdot}L_2 = \{(\mathtt{h}_1,\mathtt{u}_1\mathtt{u}_2) \mid (\mathtt{h}_1,\mathtt{u}_1) \in L_1 \text{ and } (\mathtt{h}_1\mathtt{u}_1,\mathtt{u}_2) \in L_2\}$
– $L_1{\circ}L_2 = \{(\mathtt{h}_1,\mathtt{v}_1\mathtt{a}\mathtt{v}_2) \mid \exists\tilde{\gamma} \text{ such that } (\mathtt{h}_1,\mathtt{v}_1(\mathtt{a}|_\sigma,\mathtt{a}|_\gamma,\tilde{\gamma})) \in L_1 \text{ and } (\mathtt{h}_1\mathtt{v}_1,(\mathtt{a}|_\sigma,\tilde{\gamma},\mathtt{a}|_{\gamma'})\mathtt{v}_2) \in L_2\}$

We use $\mathbb{\lambda}$ to denote the set $\{(\mathtt{h},\epsilon) \mid \mathtt{h} \in \Upsilon^*\}$ where ϵ denotes the empty word. For a language L of h-words we use L^0 to denote the set $\mathbb{\lambda}$. We use L^i to denote $L^{i-1}{\cdot}L$. We use L^* and L^+ to denote $\bigcup_{i\geq 0} L^i$ and $\bigcup_{i>0} L^i$, respectively. We use L^ω for the language composed of infinitely many concatenations of L with itself. We use L^∞ for the union $L^* \cup L^\omega$.

For a finite/infinite h-word $\mathfrak{w} = (\mathtt{h},\mathtt{w})$ where $\mathtt{w} = \mathtt{a}_0\mathtt{a}_1\cdots\mathtt{a}_n$ or $\mathtt{w} = \mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\cdots$ and integers j,k we use $\mathfrak{w}^i$ to denote the letter $(i{+}1)^{th}$ letter of $\mathtt{w}$. That is, $\mathfrak{w}^i = \mathtt{a}^i$. We use $\mathfrak{w}^{i\cdot\cdot}$ to denote the suffix of $\mathfrak{w}$ starting at $\mathfrak{w}^i$. That is, $\mathfrak{w}^{i\cdot\cdot} = (\mathtt{h}\mathtt{a}_0\mathtt{a}_1\cdots\mathtt{a}_{i-1},\ \mathtt{a}_i\mathtt{a}_{i+1}\cdots\mathtt{a}_n)$ or $\mathfrak{w}^{i\cdot\cdot} = (\mathtt{h}\mathtt{a}_0\mathtt{a}_1\cdots\mathtt{a}_{i-1},\ \mathtt{a}_i\mathtt{a}_{i+1}\cdots)$. We use $\mathfrak{w}^{i..j}$ to denote the finite word starting at position i and ending at position j. That is, $\mathfrak{w}^{i..j} = (\mathtt{h}\mathtt{a}_0\mathtt{a}_1\cdots\mathtt{a}_{i-1},\ \mathtt{a}_i\mathtt{a}_{i+1}\cdots\mathtt{a}_j)$. We make use of an "overflow" and "underflow" for the positions of $\mathfrak{w}$. That is, $\mathfrak{w}^{j\cdot\cdot}$ and $\mathfrak{w}^{j..k}$ are defined for j bigger than the last position of $\mathfrak{w}$ and for $k < j$ as follows: $\mathfrak{w}^{j\cdot\cdot} = \mathfrak{w}^{j..k} = (\mathtt{h}\mathtt{w},\epsilon)$.

B.3.1.2.4 The Full Semantics

For a (Boolean) expression $e = d$, the Boolean semantics of $e = d$ with respect to a given set of local variables Y and a clock context κ, denoted $\mathcal{B}_{\kappa,\mathrm{Y}}(e)$, is the language of all h-letters $(\mathtt{h},\mathtt{a})$ on which $e = d$ holds. It is defined formally as follows. The definition is given by mutual induction with the language of SEREs, denoted $\mathcal{L}_{\kappa,\mathrm{Y}}(\cdot)$, given in Definition 20 and the definition of ticks given in Definition 19.

Definition 14 (The Boolean Semantics)

- *Base cases:*
  - $\mathcal{B}_{\kappa,\mathrm{Y}}(p{=}d) = \{(\mathbb{h},\mathbb{a}) \mid \mathbb{a}|_\sigma(p) = d\}$
  - $\mathcal{B}_{\kappa,\mathrm{Y}}(y{=}d) = \{(\mathbb{h},\mathbb{a}) \mid \mathbb{a}|_\gamma(y) = d\}$
- *Standard operators:*
  - $\mathcal{B}_{\kappa,\mathrm{Y}}(\odot e{=}d) = \{(\mathbb{h},\mathbb{a}) \mid \exists d' \in \mathcal{D}\ s.t.\ \odot(d'){=}d\ and\ (\mathbb{h},\mathbb{a}){\in}\mathcal{B}_{\kappa,\mathrm{Y}}(e{=}d')\}$
  - $\mathcal{B}_{\kappa,\mathrm{Y}}(e_1 \otimes e_2{=}d) = \left\{(\mathbb{h},\mathbb{a}) \;\middle|\; \begin{array}{l}\exists d_1,d_2 \in \mathcal{D}\ s.t.\ (d_1 \otimes d_2){=}d\ and\\ (\mathbb{h},\mathbb{a}){\in}\mathcal{B}_{\kappa,\mathrm{Y}}(e_1{=}d_1)\ and\ (\mathbb{h},\mathbb{a}){\in}\mathcal{B}_{\kappa,\mathrm{Y}}(e_2{=}d_2)\end{array}\right\}$
- *Past expressions:*
  - $\mathcal{B}_{\kappa,\mathrm{Y}}(prev(e,n){=}d) = \left\{(\mathbb{h},\mathbb{a}) \;\middle|\; \begin{array}{l}\exists \mathbb{u},\mathbb{v} \in \Upsilon^*\ \exists \mathbb{b} \in \Upsilon\ s.t.\ \mathbb{h}\mathbb{a} = \mathbb{u}\mathbb{b}\mathbb{v}\ and\\ (\mathbb{u},\mathbb{b}\mathbb{v}) \in past\_tick^{n+1}_{\kappa,\mathrm{Y}}\ and\ (\mathbb{u},\mathbb{b}) \in \mathcal{B}_{\kappa,\mathrm{Y}}(e{=}d)\end{array}\right\}$
  - $\mathcal{B}_{\kappa,\mathrm{Y}}(ended(r){=}\mathrm{T}) = \{(\mathbb{h},\mathbb{a}) \mid \exists \mathbb{u},\mathbb{v} \in \Upsilon^*\ s.t.\ \mathbb{h} = \mathbb{u}\mathbb{v}\ and\ (\mathbb{u},\mathbb{v}\mathbb{a}) \in \mathcal{L}_{\kappa,\mathrm{Y}}(r)\}$

Definition 15 (The value of an expression) *For an h-letter* $(\mathbb{h},\mathbb{a})$ *and an expression* e *we use* $e_{\kappa,\mathrm{Y}}(\mathbb{h},\mathbb{a})$ *to denote the single element of the language* $\{d \mid (\mathbb{h},\mathbb{a}) \in \mathcal{B}_{\kappa,\mathrm{Y}}(e{=}d)\}$.

Definition 16 (Agrees Relative to Y) *Let* $\gamma_1,\gamma_2 \in \Gamma$ *and* $\mathrm{Y} \subseteq \mathcal{V}$. *We say that* γ_1 agrees *with* γ_2 *relative to* Y, *denoted* $\gamma_1 \overset{\mathrm{Y}}{\sim} \gamma_2$, *if for every* $y \in \mathrm{Y}$ *we have* $\gamma_1(y) = \gamma_2(y)$.

Given a sequence of assignments Z←E where $\mathrm{Z} = \{z_1, z_2, \ldots, z_n\}$ is a finite (possibly empty) set of local variables and $\mathrm{E} = \{e_1, e_2 \ldots, e_n\}$ is a sequence of assignments of the same length, we would like to compute the results of all these assignments taking place one after the other. That is, first the assignment $z_1{\leftarrow}e_1$ takes place, then the assignment $z_2{\leftarrow}e_2$ takes place etc., until the assignment $z_n{\leftarrow}e_n$ takes place.

Definition 17 (Sequence of Assignments) *Let* κ *be a Boolean expression and* $\mathrm{Y} \subseteq \mathcal{V}$. *Let* z *be a local variable and* e *an expression. Let* $\mathrm{Z} = z_1,\ldots,z_n$ *be a sequence of local variables and* E *a sequence of expressions* $\mathrm{E} = e_1,\ldots,e_n$ *of the same length.*

- *We write* $[z{\leftarrow}e]_{\kappa,\mathrm{Y}}(\mathbb{h},\mathbb{a})$ *to denote the valuation* $\hat{\gamma}$ *such that* $\hat{\gamma}(z) = e_{\kappa,\mathrm{Y}}(\mathbb{h},\mathbb{a})$ *and for every local variable* $y \in \mathcal{V}\backslash\{z\}$ *we have that* $\hat{\gamma}(y) = \gamma(y)$.
- *We write* $[z_1{\leftarrow}e_1,\ldots,z_n{\leftarrow}e_n]_{\kappa,\mathrm{Y}}(\mathbb{h},\mathbb{a})$ *to denote the recursive application*

$$[z_2{\leftarrow}e_2,\ldots,z_n{\leftarrow}e_n]_{\kappa,\mathrm{Y}}(\mathbb{h},\langle\mathbb{a}|_\sigma,\hat{\gamma},\mathbb{a}|_{\gamma'}\rangle)$$

  where $\hat{\gamma} = [z_1{\leftarrow}e_1]_{\kappa,\mathrm{Y}}(\mathbb{h},\mathbb{a})$.

We write Z←E *to abbreviate* $z_1{\leftarrow}e_1,\ldots,z_n{\leftarrow}e_n$. *If* Z *is empty we read* Z←E *as the empty assignment* $\_\_$ *and understand* $[\mathrm{Z}{\leftarrow}\mathrm{E}]_{\kappa,\mathrm{Y}}(\mathbb{h},\mathbb{a})$ *to be* $\mathbb{a}|_\gamma$.

The *assignment language* of Z←E formally defined below returns the set of all h-letters $(\mathbb{h},\mathbb{a})$ such that $\mathbb{a}$ records correctly the assignment Z←E taking place at on $\mathbb{a}$ given the past is $\mathbb{h}$.

Definition 18 (The Assignment Language) *Let* κ *be a Boolean expression and* $\mathrm{Y} \subseteq \mathcal{V}$. *Let* Z *be a sequence of local variables and* E *a sequence of expressions* E *of the same length. The assignment language of* Z←E *with respect to* κ, Y *is defined as follows.*

- $\mathcal{A}_{\kappa,\mathrm{Y}}(\mathrm{Z}{\leftarrow}\mathrm{E}) = \{(\mathbb{h},\mathbb{a}) \mid \mathbb{a}|_{\gamma'} \overset{\mathrm{Y}}{\sim} [\mathrm{Z}{\leftarrow}\mathrm{E}]_{\kappa,\mathrm{Y}}(\mathbb{h},\mathbb{a})\}$

Definition 19 (no_tick,tick,past_tick) *Let* κ *be a Boolean expression and* $\mathrm{Y} \subseteq \mathcal{V}$. *We define the languages* $no\_tick_{\kappa,\mathrm{Y}}$, $tick_{\kappa,\mathrm{Y}}$ *and* $past\_tick_{\kappa,\mathrm{Y}}$ *as follows:*

- $no\_tick_{\kappa,\mathrm{Y}} = (\mathcal{B}_{\mathit{true},\mathrm{Y}}(\neg\kappa) \cap \mathcal{A}_{\kappa,\mathrm{Y}}(\_\_))$
- $tick_{\kappa,\mathrm{Y}} = (no\_tick_{\kappa,\mathrm{Y}})^*{\cdot}\mathcal{B}_{\mathit{true},\mathrm{Y}}(\kappa)$

- $past\_tick_{\kappa,\mathrm{Y}} = \mathcal{B}_{\mathsf{true},\mathrm{Y}}(\kappa)\cdot(no\_tick_{\kappa,\mathrm{Y}})^*$

For a regular expression r the *language* of r with respect to κ, Y, denoted $\mathcal{L}_{\kappa,\mathrm{Y}}(r)$, is defined by mutual induction with Definition 14 as follows.

Definition 20 (The Language of SEREs)

- *Base cases:*
  - $\mathcal{L}_{\kappa,\mathrm{Y}}([*0]) = \mathbb{\lambda}$
  - $\mathcal{L}_{\kappa,\mathrm{Y}}(e{=}d, \mathrm{Z}{\leftarrow}\mathrm{E}) = tick_{\kappa,\mathrm{Y}} \cap (\Upsilon^*\cdot(\mathcal{B}_{\kappa,\mathrm{Y}}(e{=}d) \cap \mathcal{A}_{\kappa,\mathrm{Y}}(\mathrm{Z}{\leftarrow}\mathrm{E})))$
- *Standard SERE operators:*
  - $\mathcal{L}_{\kappa,\mathrm{Y}}(\{r\}) = \mathcal{L}_{\kappa,\mathrm{Y}}(r)$
  - $\mathcal{L}_{\kappa,\mathrm{Y}}(r_1{\cdot}r_2) = \mathcal{L}_{\kappa,\mathrm{Y}}(r_1){\cdot}\mathcal{L}_{\kappa,\mathrm{Y}}(r_2)$
  - $\mathcal{L}_{\kappa,\mathrm{Y}}(r_1{\circ}r_2) = \mathcal{L}_{\kappa,\mathrm{Y}}(r_1){\circ}\mathcal{L}_{\kappa,\mathrm{Y}}(r_2)$
  - $\mathcal{L}_{\kappa,\mathrm{Y}}(r[+]) = \mathcal{L}_{\kappa,\mathrm{Y}}(r)^+$
  - $\mathcal{L}_{\kappa,\mathrm{Y}}(r_1 \ \cup \ r_2) = \mathcal{L}_{\kappa,\mathrm{Y}}(r_1) \cup \mathcal{L}_{\kappa,\mathrm{Y}}(r_2)$
  - $\mathcal{L}_{\kappa,\mathrm{Y}}(r_1 \ \cap \ r_2) = \mathcal{L}_{\kappa,\mathrm{Y}}(r_1) \cap \mathcal{L}_{\kappa,\mathrm{Y}}(r_2)$
- *Context operators:*
  - $\mathcal{L}_{\kappa,\mathrm{Y}}(\{\mathsf{var}(\mathrm{Z})\ r\}) = \mathcal{L}_{\kappa,\mathrm{Y}\cup\mathrm{Z}}(r)$
  - $\mathcal{L}_{\kappa,\mathrm{Y}}(\{\mathsf{free}(\mathrm{Z})\ r\}) = \mathcal{L}_{\kappa,\mathrm{Y}\setminus\mathrm{Z}}(r)$
  - $\mathcal{L}_{\kappa,\mathrm{Y}}(r@b) = \mathcal{L}_{b,\mathrm{Y}}(r)$

Definition 21 (The Language of Proper Prefixes of SEREs)

- *Base cases:*
  - $\mathcal{F}_{\kappa,\mathrm{Y}}([*0]) = \emptyset$
  - $\mathcal{F}_{\kappa,\mathrm{Y}}(e = d, \mathrm{Z}{\leftarrow}\mathrm{E}) = (no\_tick_{\kappa,\mathrm{Y}})^*$
- *Standard SERE operators:*
  - $\mathcal{F}_{\kappa,\mathrm{Y}}(\{r\}) = \mathcal{F}_{\kappa,\mathrm{Y}}(r)$
  - $\mathcal{F}_{\kappa,\mathrm{Y}}(r_1{\cdot}r_2) = \mathcal{F}_{\kappa,\mathrm{Y}}(r_1) \cup (\mathcal{L}_{\kappa,\mathrm{Y}}(r_1){\cdot}\mathcal{F}_{\kappa,\mathrm{Y}}(r_2))$
  - $\mathcal{F}_{\kappa,\mathrm{Y}}(r_1{\circ}r_2) = \mathcal{F}_{\kappa,\mathrm{Y}}(r_1) \cup (\mathcal{L}_{\kappa,\mathrm{Y}}(r_1){\circ}\mathcal{F}_{\kappa,\mathrm{Y}}(r_2))$
  - $\mathcal{F}_{\kappa,\mathrm{Y}}(r[+]) = \mathcal{L}_{\kappa,\mathrm{Y}}(r)^*{\cdot}\mathcal{F}_{\kappa,\mathrm{Y}}(r)$
  - $\mathcal{F}_{\kappa,\mathrm{Y}}(r_1 \ \cup \ r_2) = \mathcal{F}_{\kappa,\mathrm{Y}}(r_1) \cup \mathcal{F}_{\kappa,\mathrm{Y}}(r_2)$
  - $\mathcal{F}_{\kappa,\mathrm{Y}}(r_1 \ \cap \ r_2) = \mathcal{F}_{\kappa,\mathrm{Y}}(r_1) \cap \mathcal{F}_{\kappa,\mathrm{Y}}(r_2)$
- *Context operators:*
  - $\mathcal{F}_{\kappa,\mathrm{Y}}(\{\mathsf{var}(\mathrm{Z})\ r\}) = \mathcal{F}_{\kappa,\mathrm{Y}\cup\mathrm{Z}}(r)$
  - $\mathcal{F}_{\kappa,\mathrm{Y}}(\{\mathsf{free}(\mathrm{Z})\ r\}) = \mathcal{F}_{\kappa,\mathrm{Y}\setminus\mathrm{Z}}(r)$
  - $\mathcal{F}_{\kappa,\mathrm{Y}}(r@b) = \mathcal{F}_{b,\mathrm{Y}}(r)$

Definition 22 (The Loop Language of SEREs)

- *Base cases:*
  - $\mathcal{I}_{\kappa,\mathrm{Y}}([*0]) = \emptyset$
  - $\mathcal{I}_{\kappa,\mathrm{Y}}(e{=}d, \mathrm{Z}{\leftarrow}\mathrm{E}) = (no\_tick_{\kappa,\mathrm{Y}})^\omega$
- *Standard SERE operators:*
  - $\mathcal{I}_{\kappa,\mathrm{Y}}(\{r\}) = \mathcal{I}_{\kappa,\mathrm{Y}}(r)$
  - $\mathcal{I}_{\kappa,\mathrm{Y}}(r_1{\cdot}r_2) = \mathcal{I}_{\kappa,\mathrm{Y}}(r_1) \cup (\mathcal{L}_{\kappa,\mathrm{Y}}(r_1){\cdot}\mathcal{I}_{\kappa,\mathrm{Y}}(r_2))$
  - $\mathcal{I}_{\kappa,\mathrm{Y}}(r_1{\circ}r_2) = \mathcal{I}_{\kappa,\mathrm{Y}}(r_1) \cup (\mathcal{L}_{\kappa,\mathrm{Y}}(r_1){\circ}\mathcal{I}_{\kappa,\mathrm{Y}}(r_2))$
  - $\mathcal{I}_{\kappa,\mathrm{Y}}(r[+]) = (\mathcal{L}_{\kappa,\mathrm{Y}}(r)^*{\cdot}\mathcal{I}_{\kappa,\mathrm{Y}}(r)) \cup \mathcal{L}_{\kappa,\mathrm{Y}}(r)^\omega$
  - $\mathcal{I}_{\kappa,\mathrm{Y}}(r_1 \ \cup \ r_2) = \mathcal{I}_{\kappa,\mathrm{Y}}(r_1) \cup \mathcal{I}_{\kappa,\mathrm{Y}}(r_2)$
  - $\mathcal{I}_{\kappa,\mathrm{Y}}(r_1 \ \cap \ r_2) = \mathcal{I}_{\kappa,\mathrm{Y}}(r_1) \cap \mathcal{I}_{\kappa,\mathrm{Y}}(r_2)$
- *Context operators:*
  - $\mathcal{I}_{\kappa,\mathrm{Y}}(\{\mathsf{var}(\mathrm{Z})\ r\}) = \mathcal{I}_{\kappa,\mathrm{Y}\cup\mathrm{Z}}(r)$
  - $\mathcal{I}_{\kappa,\mathrm{Y}}(\{\mathsf{free}(\mathrm{Z})\ r\}) = \mathcal{I}_{\kappa,\mathrm{Y}\setminus\mathrm{Z}}(r)$
  - $\mathcal{I}_{\kappa,\mathrm{Y}}(r@b) = \mathcal{I}_{b,\mathrm{Y}}(r)$

We define the semantics of a formula with respect to an h-word $\mathfrak{w}$ and a tuple $\tau = \langle \kappa, \mathrm{Y}, \upsilon \rangle$ where κ is a Boolean expression indicating the clock context, $\mathrm{Y} \subseteq \mathcal{V}$ is a set of local variables indicating the controlled variables, and $\upsilon \in \{\mathrm{S}, \mathrm{N}, \mathrm{W}\}$ denotes the view (where $\mathrm{S}, \mathrm{N}, \mathrm{W}$ correspond to the *strong*, *neutral* and *weak* views, respectively). For a view υ we use $\overline{\upsilon}$ to denote the *dual view* which is S if $\upsilon = \mathrm{W}$, W if $\upsilon = \mathrm{S}$, and υ otherwise (if $\upsilon = \mathrm{N}$). We use j and k below to denote non-negative integers.

Intuitively, the semantics of local variables are such that an LTL operator does not change the value of a local variable. For example, in the formula $\mathsf{G}\,(var(z)\ \{req, z{\leftarrow}tag\} \mapsto \mathsf{X!}\,\mathsf{F}(ack{\wedge}tag = z))$, the value of z "seen" by the operand of the F operator should be the value assigned by the left-hand side of the suffix implication (exactly as in the following alternative formulation of the same property: $\mathsf{G}\,(var(z)\ \{req, z{\leftarrow}tag\} \mapsto \{\mathsf{true}{\cdot}\mathsf{true}[*]{\cdot}ack \wedge tag = z\}!)$). Conceptually, this means that a formula should "see" the value of the local variables as they were at the beginning of the word, unless a SERE is encountered, in which case the values "seen" should be those dictated by the assignments made within the SERE. We accomplish this as follows: The semantics of $w \models \varphi$ for a word w over Σ takes an initial context of local variables and holds it constant across an extended word $\mathbb{w}$ whose σ component is exactly w. The semantics of $r!$, r and $r \mapsto \varphi$ then release the constant value of local variables at the points on the word where a "match" of r is being checked. This allows the local variables to be controlled by r. Finally, in the case of $r \mapsto \varphi$, "after" a match of r has been "seen", the semantics constrains the values of the local variables to take on constant values starting from the end of the "match" of the SERE. We accomplish this using three auxiliary definitions, below.

Definition 23 (Good) *We say that* $\mathbb{w}$ *is* good *if* $\mathbb{w} = \epsilon$ *or* $\mathbb{w} = \mathbb{a}$ *or* $\mathbb{w} = \mathbb{a}\mathbb{b}\mathbb{v}$ *and* $\mathbb{b}\mathbb{v}$ *is good and* $\mathbb{b}|_\gamma = \mathbb{a}|'_\gamma$*, that is, the pre-value of the local variables on a letter is the same as the post-value on the previous letter (i.e. is the result of the assignments that took place on the previous letter).*

Definition 24 (The equivalence class of $\mathbb{o}$ releasing γ) *Let* $\mathbb{o} = (\mathbb{h}, \mathbb{w})$*. Then* $[\![\mathbb{o}]\!]$ *denotes the equivalence class* $\{(\mathbb{h}', \mathbb{w}') \mid \mathbb{h}' = \mathbb{h}$*,* $\mathbb{w}'$ *is good,* $\mathbb{w}'|_\sigma = \mathbb{w}|_\sigma$ *and either* $\mathbb{w} = \mathbb{w}' = \epsilon$ *or* $\mathbb{w}^0|_\gamma = \mathbb{w}'^0|_\gamma\}$*.*

Definition 25 (Constraining γ on $\mathbb{o}$) *Let* $\mathbb{o} = (\mathbb{h}, \mathbb{w})$ *and* $\hat{\gamma} \in \Gamma$*. Then* $\mathbb{o}[(\gamma{,}\gamma') \leftarrow \hat{\gamma}]$ *denotes the word* $\mathbb{o}' = (\mathbb{h}', \mathbb{w}')$ *such that* $\mathbb{h}' = \mathbb{h}$*,* $\mathbb{w}'|_\sigma = \mathbb{w}|_\sigma$ *and* $\mathbb{w}'|_\gamma = \mathbb{w}'|_{\gamma'} \in \{\hat{\gamma}\}^\infty$*.*

The relation $\mathbb{o} \models^\tau \varphi$, read "$\mathbb{o}$ models φ in the context of τ", is formally defined as follows.

Definition 26 (Satisfaction on h-words)

I. $\mathbb{o} \models^\tau r! \iff \exists \mathbb{ß} \in [\![\mathbb{o}]\!]$ *s.t. either* $\exists j$ *s.t.* $\mathbb{ß}^{0..j} \in \mathcal{L}_{\kappa,\mathrm{Y}}(r)$ *or* $(\upsilon = \mathrm{W}$ *and* $\mathbb{ß} \in \mathcal{F}_{\kappa,\mathrm{Y}}(r) \cup \mathbb{\lambda})$

II. $\mathbb{o} \models^\tau r \iff \exists \mathbb{ß} \in [\![\mathbb{o}]\!]$ *s.t. either* $\exists j$ *s.t.* $\mathbb{ß}^{0..j} \in \mathcal{L}_{\kappa,\mathrm{Y}}(r)$ *or* $\mathbb{ß} \in \mathcal{I}_{\kappa,\mathrm{Y}}(r)$ *or* $(\upsilon \in \{\mathrm{W}, \mathrm{N}\}$ *and* $\mathbb{ß} \in \mathcal{F}_{\kappa,\mathrm{Y}}(r) \cup \mathbb{\lambda})$

III. $\mathbb{o} \models^\tau r \mapsto \varphi \iff \forall j\ \forall \mathbb{ß} \in [\![\mathbb{o}]\!]$ *s.t.* $\mathbb{ß}^{0..j} \in \mathcal{L}_{\kappa,\mathrm{Y}}(r)$ *we have that* $\mathbb{ß}^{j..}[(\gamma{,}\gamma') \leftarrow \mathbb{ß}^j|_{\gamma'}] \models^\tau \varphi$ *and either* $\upsilon \in \{\mathrm{W}, \mathrm{N}\}$ *or* $w \notin \mathcal{F}_{\kappa,\mathrm{Y}}(r) \cup \{\epsilon\}$

IV. $\mathbb{o} \models^\tau (\varphi) \iff \mathbb{o} \models^\tau \varphi$

V. $\mathbb{o} \models^\tau \neg\varphi \iff \mathbb{o} \models^{\overline{\tau}} \varphi$ *where* $\overline{\tau} = \tau[\upsilon{\leftarrow}\overline{\upsilon}]$

VI. $\mathbb{o} \models^\tau \varphi \wedge \psi \iff \mathbb{o} \models^\tau \varphi$ *and* $\mathbb{o} \models^\tau \psi$

VII. $\mathbb{o} \models^\tau \mathsf{X!}\,[k]\,\varphi \iff$ *either* $\exists j$ *s.t.* $\mathbb{o}^{0..j} \in \mathcal{L}_{\kappa,\mathrm{Y}}(tick\,[k{+}1])$ *and* $\mathbb{o}^{j..} \models^\upsilon \varphi$ *or* $(\upsilon = \mathrm{W}$ *and* $\mathbb{o} \in \mathcal{F}_{\kappa,\mathrm{Y}}(tick\,[k{+}1]) \cup \mathbb{\lambda})$

VIII. $\mathbb{o} \models^\tau \varphi \mathsf{U} \psi \iff \exists k$ *s.t.* $\mathbb{o} \models^\tau \mathsf{X!}\,[k]\,\psi$ *and* $\forall j < k$, $\mathbb{o} \models^\tau \mathsf{X!}\,[j]\,\varphi$

IX. $\mathbb{o} \models^\tau \varphi\ \mathsf{sync\_abort}\ t \iff$ *either* $\mathbb{o} \models^\tau \varphi$ *or* $\exists k$ *s.t.* $\mathbb{o}^{k..} \models^{\tau'} ended(t)$ *and* $\mathbb{o}^{0..k-1} \models^{\tau''} \varphi$ *where* $\tau' = \tau[\upsilon{\leftarrow}\mathrm{S}]$ *and* $\tau'' = \tau[\upsilon{\leftarrow}\mathrm{W}]$

X. $\mathbb{o} \models^\tau (\mathsf{var}(\mathrm{Z})\ \varphi) \iff \mathbb{o} \models^{\tau'} \varphi$ *where* $\tau' = \tau[\mathrm{Y}{\leftarrow}\mathrm{Y} \cup \mathrm{Z}]$

XI. $\mathbb{o} \models^\tau \varphi @ b \iff \mathbb{o} \models^{\tau'} \varphi$ *where* $\tau' = \tau[\kappa{\leftarrow}b]$

Definition 27 (Satisfaction over Σ) *Let* w *be a word over* Σ *and* φ *be an FL formula. We say that* w models φ *in the view* υ*, denoted* $w \models^\upsilon \varphi$ *iff for all* $\hat{\gamma} \in \Gamma$ *and all extended words* $\mathbb{w}$ *such that* $\mathbb{w}|_\sigma = w$ *we have that* $(\epsilon, \mathbb{w})[(\gamma{,}\gamma') \leftarrow \hat{\gamma}] \models^{\tau_\upsilon} \varphi$*, where* $\tau_\upsilon = \langle \mathsf{true}, \emptyset, \upsilon\rangle$*. That is, the past is empty, the pre- and post-values are* γ *on every letter, the initial clock context is* true *and the initial set of controlled variables is empty.*

Definition 28 (Holds Weakly, Strong, Neutrally, Pending, Fails) *Let w be a word over Σ and φ an FL formula. We say that*

- φ holds weakly *on w if $w \models^{\text{W}} \varphi$*
- φ holds (neutrally) *on w if $w \models^{\text{N}} \varphi$*
- φ holds strongly *on w if $w \models^{\text{S}} \varphi$*
- φ *is* pending *on word w iff $w \models^{\text{W}} \varphi$ and $w \not\models^{\text{N}} \varphi$*
- φ fails *on word w iff $w \not\models^{\text{W}} \varphi$*

B.3.2 Semantics of OBE formulas

The semantics of OBE formulas are defined over states in the *model*, rather than finite or infinite words. Let f be an OBE formula, $M = (S, S_0, R, \mathcal{P}, L)$ a model and $s \in S$ a state of the model. The notation $M, s \models f$ means that f holds in state s of model M. The notation $M \models f$ is equivalent to $\forall s \in S_0 : M, s \models f$. In other words, f is valid for every initial state of M.

The semantics of OBE formulas are defined inductively, using as the base case the semantics of *Basic Boolean expressions* over *letters* in $2^{\mathcal{P}}$, as given in Definition 7.

The semantics of an OBE formula are those of standard CTL. The semantics are defined as follows, where b denotes a Boolean expression and f, f_1, and f_2 denote OBE formulas.

I. $M, s \models b \iff L(s) \in \mathcal{B}(b = \text{T})$

II. $M, s \models (f) \iff M, s \models f$

III. $M, s \models \neg f \iff M, s \not\models f$

IV. $M, s \models f_1 \wedge f_2 \iff M, s \models f_1$ and $M, s \models f_2$

V. $M, s \models \mathsf{EX}\ f \iff$ there exists a computation path π of M such that $|\pi| > 1$, $\pi_0 = s$, and $M, \pi_1 \models f$

VI. $M, s \models \mathsf{E}[f_1\ \mathsf{U}\ f_2] \iff$ there exists a computation path π of M such that $\pi_0 = s$ and there exists $k < |\pi|$ such that $M, \pi_k \models f_2$ and for every j such that $j < k$: $M, \pi_j \models f_1$

VII. $M, s \models \mathsf{EG}\ f \iff$ there exists a computation path π of M such that $\pi_0 = s$ and for every j such that $0 \leq j < |\pi|$: $M, \pi_j \models f$

B.4 Syntactic Sugaring

The remainder of the temporal layer is syntactic sugar. In other words, it does not add expressive power, and every piece of syntactic sugar can be defined in terms of the basic operators presented above. The syntactic sugar is defined below.

NOTE –
The definitions given here do not necessarily represent the most efficient implementation. In some cases, there is an equivalent syntactic sugaring, or a direct implementation, that is more efficient.

B.4.1 Additional Boolean expressions

Let $b \in \mathcal{B}$ and $e \in \mathcal{E}$. Then additional Boolean expressions can be viewed as abbreviations of the core Boolean expressions given in Definition 1, as follows:

- $rose(b) \stackrel{\text{def}}{=} \neg prev(b) \wedge b$
- $fell(b) \stackrel{\text{def}}{=} prev(b) \wedge \neg b$
- $stable(e) \stackrel{\text{def}}{=} prev(e) = e$

B.4.2 Additional expressions

Let $e \in \mathcal{E}$. Then additional expressions can be viewed as abbreviations of the expressions given in Definition 2, as follows:

- $prev(e) \stackrel{\text{def}}{=} prev(e, 1)$

B.4.3 Additional SEREs

We regard a *procedural block* as mechanism for both calculating a set of expressions and assigning them to a set of local variables which are given to the procedural block as inputs. Formally,

- $f(z_1, \ldots, z_n)$ where f is a procedural block and $z_1, \ldots, z_n$ are local variables is interpreted as an abbreviation for the sequence of assignments $z_1 \leftarrow e_{f_1}, \ldots, z_n \leftarrow e_{f_n}$ where $e_{f_1}, \ldots, e_{f_n}$ are expressions corresponding to intermediate calculations of f.

Let $b, c \in \mathcal{B}$ and $e \in \mathcal{E}$. Let k be a non-negative integer. Then the following are SEREs:

- $rose(b, c) \stackrel{\text{def}}{=} rose(b)@c$
- $fell(b, c) \stackrel{\text{def}}{=} fell(b)@c$
- $stable(e, c) \stackrel{\text{def}}{=} stable(e)@c$
- $prev(e, k, c) \stackrel{\text{def}}{=} prev(e, k)@c$
- $ended(r, c) \stackrel{\text{def}}{=} ended(r)@c$

B.4.4 Additional SERE operators

Let i, j, k, and l be integer constants such that $i \geq 0$, $j \geq i$, $k \geq 1$, $l \geq k$. Then additional SERE operators can be viewed as abbreviations of the core SERE operators given in Definition 3, as follows, where b denotes a boolean expression, and r denotes a SERE.

- $r[*] \stackrel{\text{def}}{=} [*0] \cup r[+]$
- $r[*0] \stackrel{\text{def}}{=} [*0]$
- $r[*k] \stackrel{\text{def}}{=} \overbrace{r \cdot r \cdot \ldots \cdot r}^{k \text{ times}}$
- $r[*i..j] \stackrel{\text{def}}{=} r[*i] \mid \ldots \mid r[*j]$
- $r[*i..] \stackrel{\text{def}}{=} r[*i] \cdot r[*]$
- $r[*..i] \stackrel{\text{def}}{=} r[*0] \mid \ldots \mid r[*i]$

- $r[*..] \stackrel{\text{def}}{=} r[*0..]$
- $[+] \stackrel{\text{def}}{=} \mathsf{true}[+]$
- $[*] \stackrel{\text{def}}{=} \mathsf{true}[*]$
- $[*i] \stackrel{\text{def}}{=} \mathsf{true}[*i]$
- $[*i..j] \stackrel{\text{def}}{=} \mathsf{true}[*i..j]$
- $[*i..] \stackrel{\text{def}}{=} \mathsf{true}[*i..]$
- $[*..i] \stackrel{\text{def}}{=} \mathsf{true}[*..i]$
- $[*..] \stackrel{\text{def}}{=} \mathsf{true}[*..]$
- $b[= i] \stackrel{\text{def}}{=} \{\neg b[*]\cdot b\}[*i]\cdot\neg b[*]$
- $b[= i..j] \stackrel{\text{def}}{=} b[= i] \mid ... \mid b[= j]$
- $b[= i..] \stackrel{\text{def}}{=} b[= i]\cdot[*]$
- $b[= ..i] \stackrel{\text{def}}{=} b[= 0] \mid ... \mid b[= i]$
- $b[= ..] \stackrel{\text{def}}{=} b[= 0..]$
- $b[\rightarrow] \stackrel{\text{def}}{=} \neg b[*]\cdot b$
- $b[\rightarrow k] \stackrel{\text{def}}{=} \{\neg b[*]\cdot b\}[*k]$
- $b[\rightarrow k..l] \stackrel{\text{def}}{=} b[\rightarrow k] \mid ... \mid b[\rightarrow l]$
- $b[\rightarrow k..] \stackrel{\text{def}}{=} b[\rightarrow k] \mid \{b[\rightarrow k]\cdot[*]\cdot b\}$
- $b[\rightarrow ..k] \stackrel{\text{def}}{=} b[\rightarrow 1] \mid ... \mid b[\rightarrow k]$
- $b[\rightarrow ..] \stackrel{\text{def}}{=} b[\rightarrow 1..]$
- $\mathsf{skip} \stackrel{\text{def}}{=} \{\mathsf{free}(\mathcal{V})\ \mathsf{true}\}$ where $\mathcal{V}$ is the entire set of local variables
- $r_1\ \&\ r_2 \stackrel{\text{def}}{=} \{\ \{r_1\cdot\mathsf{skip}[*]\} \cap r_2\ \} \cup \{\ r_1 \cap \{r_2\cdot\mathsf{skip}[*]\}\ \}$
- $r_1\ within\ r_2 \stackrel{\text{def}}{=} \{\mathsf{skip}[*]\cdot\ r_1\cdot\ \mathsf{skip}[*]\}\ \cap\ \{r_2\}$
- $\{\mathsf{var}(\mathrm{Z}\leftarrow\mathrm{E})\ r\} \stackrel{\text{def}}{=} \{\mathsf{var}(\mathrm{Z})\ \{\{\mathsf{true}, \mathrm{Z} \leftarrow \mathrm{E}\}\circ r\} \cup \{[*0]\ \cap\ r\}\}$

B.4.5 Additional FL operators

Let i, j, k and l are integers such that $i \geq 0$, $j \geq i$, $k > 0$ and $l \geq k$. Then additional FL operators can be viewed as abbreviations of the core operators given in Definition 4, as follows, where b denotes a boolean expression, r, r_1, and r_2 denote SEREs, and φ, φ_1, and φ_2 denote FL formulas.

- $\varphi_1 \vee \varphi_2 \stackrel{\text{def}}{=} \neg(\neg\varphi_1 \wedge \neg\varphi_2)$
- $\varphi_1 \rightarrow \varphi_2 \stackrel{\text{def}}{=} \neg\varphi_1 \vee \varphi_2$
- $\varphi_1 \leftrightarrow \varphi_2 \stackrel{\text{def}}{=} (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$

- $\mathsf{X}[i]\varphi \stackrel{\text{def}}{=} \neg\mathsf{X}![i]\ \neg\varphi$
- $\mathsf{X}!\varphi \stackrel{\text{def}}{=} \mathsf{X}![1]\ \varphi$
- $\mathsf{X}\varphi \stackrel{\text{def}}{=} \mathsf{X}[1]\ \varphi$

- $\mathsf{F}\varphi \stackrel{\text{def}}{=} \mathsf{true}\ \mathsf{U}\ \varphi$
- $\mathsf{G}\varphi \stackrel{\text{def}}{=} \neg\mathsf{F}\neg\varphi$
- $\varphi_1\ \mathsf{W}\ \varphi_2 \stackrel{\text{def}}{=} (\varphi_1\ \mathsf{U}\ \varphi_2) \vee \mathsf{G}\varphi_1$

- $always\ \varphi \stackrel{\text{def}}{=} \mathsf{G}\ \varphi$
- $never\ \varphi \stackrel{\text{def}}{=} \mathsf{G}\ \neg\varphi$
- $next!\ \varphi \stackrel{\text{def}}{=} \mathsf{X}!\ \varphi$
- $next\ \varphi \stackrel{\text{def}}{=} \mathsf{X}\ \varphi$
- $eventually!\ \varphi \stackrel{\text{def}}{=} \mathsf{F}\varphi$

- $\varphi_1\ until!\ \varphi_2 \stackrel{\text{def}}{=} \varphi_1\ \mathsf{U}\ \varphi_2$
- $\varphi_1\ until\ \varphi_2 \stackrel{\text{def}}{=} \varphi_1\ \mathsf{W}\ \varphi_2$
- $\varphi_1\ until!\_\ \varphi_2 \stackrel{\text{def}}{=} \varphi_1\ \mathsf{U}\ (\varphi_1 \wedge \varphi_2)$
- $\varphi_1\ until\_\ \varphi_2 \stackrel{\text{def}}{=} \varphi_1\ \mathsf{W}\ (\varphi_1 \wedge \varphi_2)$

- $\varphi_1\ before!\ \varphi_2 \stackrel{\text{def}}{=} (\neg\varphi_2)\ \mathsf{U}\ (\varphi_1 \wedge \neg\varphi_2)$
- $\varphi_1\ before\ \varphi_2 \stackrel{\text{def}}{=} (\neg\varphi_2)\ \mathsf{W}\ (\varphi_1 \wedge \neg\varphi_2)$
- $\varphi_1\ before!\_\ \varphi_2 \stackrel{\text{def}}{=} (\neg\varphi_2)\ \mathsf{U}\ \varphi_1$
- $\varphi_1\ before\_\ \varphi_2 \stackrel{\text{def}}{=} (\neg\varphi_2)\ \mathsf{W}\ \varphi_1$

- $next![i]\ \varphi \stackrel{\text{def}}{=} \mathsf{X}![i]\ \varphi$
- $next[i]\ \varphi \stackrel{\text{def}}{=} \mathsf{X}[i]\ \varphi$
- $next\_a![i..j]\varphi \stackrel{\text{def}}{=} (\mathsf{X}![i]\varphi) \wedge \ldots \wedge (\mathsf{X}![j]\varphi)$
- $next\_a[i..j]\varphi \stackrel{\text{def}}{=} (\mathsf{X}[i]\varphi) \wedge \ldots \wedge (\mathsf{X}[j]\varphi)$
- $next\_e![i..j]\varphi \stackrel{\text{def}}{=} (\mathsf{X}![i]\varphi) \vee \ldots \vee (\mathsf{X}![j]\varphi)$
- $next\_e[i..j]\varphi \stackrel{\text{def}}{=} (\mathsf{X}[i]\varphi) \vee \ldots \vee (\mathsf{X}[j]\varphi)$

- $next\_event!(b)(\varphi) \stackrel{\text{def}}{=} (\neg b)\ \mathsf{U}\ b \wedge \varphi$
- $next\_event(b)(\varphi) \stackrel{\text{def}}{=} (\neg b)\ \mathsf{W}\ (b \wedge \varphi)$

- $next\_event!(b)[k](\varphi) \stackrel{\text{def}}{=} next\_event!(b)\ \overbrace{(\mathsf{X}!\ next\_event!(b)...(\mathsf{X}!\ next\_event!(b)}^{k-1\ times}(\varphi))...)$
- $next\_event(b)[k](\varphi) \stackrel{\text{def}}{=} next\_event(b)\ \overbrace{(\mathsf{X}next\_event(b)...(\mathsf{X}next\_event(b)}^{k-1\ times}(\varphi))...)$
- $next\_event\_a!(b)[k..l](\varphi) \stackrel{\text{def}}{=} next\_event!(b)[k](\varphi) \wedge ... \wedge next\_event!(b)[l](\varphi)$
- $next\_event\_a(b)[k..l](\varphi) \stackrel{\text{def}}{=} next\_event(b)[k](\varphi) \wedge ... \wedge next\_event(b)[l](\varphi)$
- $next\_event\_e!(b)[k..l](\varphi) \stackrel{\text{def}}{=} next\_event!(b)[k](\varphi) \vee ... \vee next\_event!(b)[l](\varphi)$
- $next\_event\_e(b)[k..l](\varphi) \stackrel{\text{def}}{=} next\_event(b)[k](\varphi) \vee ... \vee next\_event(b)[l](\varphi)$

- $r(\varphi) \stackrel{\text{def}}{=} r \mapsto \varphi$
- $r \Mapsto \varphi \stackrel{\text{def}}{=} \{r \circ \{\mathsf{true{\cdot}true}\}\} \mapsto \varphi$

- φ async_abort $t \stackrel{\text{def}}{=} \varphi$ sync_abort (t@true)
- φ abort $t \stackrel{\text{def}}{=} \varphi$ async_abort t

B.4.6 Parameterized SEREs and Formulas

Let r be a SERE, and l, m integers. Let S be a set of constants, integers or boolean values and p an identifier. The left hand side of the following are SEREs, equivalent to the SEREs on the right hand side.

- for p *in* S : $\cup\ r \stackrel{\text{def}}{=} \bigcup_{s \in S} \{r[p \leftarrow s]\}$
- for $p\langle l..m\rangle$ *in* S : $\cup\ r \stackrel{\text{def}}{=} \bigcup_{s_l \in S} \ldots \bigcup_{s_m \in S} \{r[p\langle l..m\rangle \leftarrow \langle s_l..s_m\rangle]\}$
- for p *in* S : $\cap\ r \stackrel{\text{def}}{=} \bigcap_{s \in S} \{r[p \leftarrow s]\}$
- for $p\langle l..m\rangle$ *in* S : $\cap\ r \stackrel{\text{def}}{=} \bigcap_{s_l \in S} \ldots \bigcap_{s_m \in S} \{r[p\langle l..m\rangle \leftarrow \langle s_l..s_m\rangle]\}$
- for p *in* S : $\&\ r \stackrel{\text{def}}{=} \&_{s \in S} \{r[p \leftarrow s]\}$
- for $\&\ p\langle l..m\rangle$ *in* S : $\&\ r \stackrel{\text{def}}{=} \&_{s_l \in S} \ldots \&_{s_m \in S} \{r[p\langle l..m\rangle \leftarrow \langle s_l..s_m\rangle]\}$

Where $r[p \leftarrow s]$ is the SERE obtained from r by replacing every occurrence of p by s and $r[p\langle l..m\rangle \leftarrow \langle s_l..s_m\rangle]$ is the SERE obtained from r by replacing every occurrence of p_j with s_j for all j such that $l \leq j \leq m$.

Let φ be an FL formula, and l, m integers. Let S be a set of constants, integers or boolean values and p an identifier. The left hand side of the following are FL formulas equivalent to the FL formulas on the right hand side.

- for p *in* $S : \vee\ \varphi \stackrel{\text{def}}{=} \bigvee_{s \in S} \varphi[p \leftarrow s]$
- for $p\langle l..m\rangle$ *in* $S : \vee\ \varphi \stackrel{\text{def}}{=} \bigvee_{s_l \in S} \ldots \bigvee_{s_m \in S} \varphi[p\langle l..m\rangle \leftarrow \langle s_l..s_m\rangle]$

- for p *in* $S : \wedge\, \varphi \stackrel{\text{def}}{=} \bigwedge_{s \in S} \varphi[p \leftarrow s]$
- for $p\langle l..m\rangle$ *in* $S : \wedge\, \varphi \stackrel{\text{def}}{=} \bigwedge_{s_l \in S} \ldots \bigwedge_{s_m \in S} \varphi[p\langle l..m\rangle \leftarrow \langle s_l..s_m\rangle]$
- forall p *in* S : $\varphi \stackrel{\text{def}}{=}$ for p *in* $S : \wedge\, \varphi$
- forall $p\langle l..m\rangle$ *in* S : $\varphi \stackrel{\text{def}}{=}$ for $p\langle l..m\rangle$ *in* $S : \wedge\, \varphi$

where $\varphi[p \leftarrow s]$ is the formula obtained from φ by replacing every occurrence of p by s and $\varphi[p\langle l..m\rangle \leftarrow \langle s_l..s_m\rangle]$ is the formula obtained from φ by replacing every occurrence of p_j with s_j for all j such that $l \le j \le m$.

B.4.7 Additional OBE operators

Let f, f_1, f_2 denote OBE formulas. Then additional OBE operators can be derived from the core OBE operators given in Definition 5 as follows:

- $f_1 \vee f_2 = \neg(\neg f_1 \wedge \neg f_2)$
- $f_1 \rightarrow f_2 = \neg f_1 \vee f_2$
- $f_1 \leftrightarrow f_2 = (f_1 \rightarrow f_2) \wedge (f_2 \rightarrow f_1)$
- $\mathsf{EF} f = \mathsf{E}[\mathsf{true\ U}\ f]$
- $\mathsf{AX} f = \neg\mathsf{EX}\neg f$
- $\mathsf{A}[f_1\ \mathsf{U}\ f_2] = \neg(\mathsf{E}[\neg f_2\ \mathsf{U}\ (\neg f_1 \wedge \neg f_2)] \vee \mathsf{EG}\neg f_2)$
- $\mathsf{AG} f = \neg\mathsf{E}[\mathsf{true\ U}\ \neg f]$
- $\mathsf{AF} f = \mathsf{A}[\mathsf{true\ U}\ f]$

B.5 Rewriting rules for clocks

In Section B.3.1.2 we gave the semantics of clocked FL formulas directly. There is an equivalent definition in terms of FL formulas without clocks, as follows: Starting from the outermost clock, use the following rules to translate clocked SEREs into unclocked SEREs, and clocked FL formulas into unclocked FL formulas.

The rewrite rules for SEREs are:

I. $\mathcal{R}^c(\{r\}) = \mathcal{R}^c(r)$

II. $\mathcal{R}^c(b) = \neg c[*] \cdot (c \wedge b)$

III. $\mathcal{R}^c(b, \mathrm{Z}{\leftarrow}\mathrm{E}) = \neg c[*] \cdot (c \wedge b, \mathrm{Z}{\leftarrow}\mathrm{E})$

IV. $\mathcal{R}^c(r_1 \cdot r_2) = \mathcal{R}^c(r_1) \cdot \mathcal{R}^c(r_2)$

V. $\mathcal{R}^c(r_1 \circ r_2) = \{\mathcal{R}^c(r_1)\} \circ \{\mathcal{R}^c(r_2)\}$

VI. $\mathcal{R}^c(r_1\ \cup\ r_2) = \{\mathcal{R}^c(r_1)\}\ \cup\ \{\mathcal{R}^c(r_2)\}$

VII. $\mathcal{R}^c(r_1\ \cap\ r_2) = \{\mathcal{R}^c(r_1)\}\ \cap\ \{\mathcal{R}^c(r_2)\}$

VIII. $\mathcal{R}^c([*0]) = [*0]$

IX. $\mathcal{R}^c(r[+]) = \{\mathcal{R}^c(r)\}[+]$

X. $\mathcal{R}^c(\mathsf{var}(\mathrm{Z})\ r) = \{\mathsf{var}(\mathrm{Z})\ \mathcal{R}^c(r)\}$

XI. $\mathcal{R}^c(\mathsf{free}(\mathrm{Z})\ r) = \{\mathsf{free}(\mathrm{Z})\ \mathcal{R}^c(r)\}$

XII. $\mathcal{R}^c(r@c_1) = \mathcal{R}^{c_1}(r)$

The rewrite rules for FL formulas are:

I. $\mathcal{F}^c((\varphi)) = (\mathcal{F}^c(\varphi))$

II. $\mathcal{F}^c(\neg\varphi) = \neg\mathcal{F}^c(\varphi)$

III. $\mathcal{F}^c(\varphi \wedge \psi) = (\mathcal{F}^c(\varphi) \wedge \mathcal{F}^c(\psi))$

IV. $\mathcal{F}^c(r!) = \mathcal{R}^c(r)!$

V. $\mathcal{F}^c(r) = \mathcal{R}^c(r)$

VI. $\mathcal{F}^c(r \mapsto \varphi) = \mathcal{R}^c(r) \mapsto \mathcal{F}^c(\varphi)$

VII. $\mathcal{F}^c(\mathsf{X!}[0]\varphi) = (\neg c\ \mathsf{U}\ (c \wedge \mathcal{F}^c(\varphi))))$
$\mathcal{F}^c(\mathsf{X!}\varphi) = (\neg c\ \mathsf{U}\ (c \wedge \mathsf{X!}\ (\neg c\ \mathsf{U}\ (c \wedge \mathcal{F}^c(\varphi)))))$
$\mathcal{F}^c(\mathsf{X!}[k]\varphi) = \mathcal{F}^c(\underbrace{\mathsf{X!}\ldots\mathsf{X!}}_{k\ times}\varphi)$ for $k \geq 2$

VIII. $\mathcal{F}^c(\varphi\ \mathsf{U}\ \psi) = ((c \rightarrow \mathcal{F}^c(\varphi))\ \mathsf{U}\ (c \wedge \mathcal{F}^c(\psi)))$

IX. $\mathcal{F}^c(\varphi\ \mathsf{sync\_abort}\ t) = \mathcal{F}^c(\varphi)\ \mathsf{sync\_abort}\ \mathcal{F}^c(t)$

X. $\mathcal{F}^c(\mathsf{var}(\mathrm{Z})\ \varphi) = (\mathsf{var}(\mathrm{Z})\ \mathcal{F}^c(\varphi))$

XI. $\mathcal{F}^c(\varphi@c_1) = \mathcal{F}^{c_1}(\varphi)$

Acknowledgments

We would like to acknowledge the contributions of the following people: Shoham Ben-David, Doron Bustan, John Havlicek, Alexander Ivrii, Erich Marschner, Johan Mårtensson, Avigail Orni, Dmitry Pidan, Sitvanit Ruah, Thomas Türk and Karen Yorav.

References

[B1] C. Eisner, D. Fisman, J. Havlicek, A. M. Yoad Lustig, and D. V. Campenhout. Reasoning with temporal logic on truncated paths. In *Proc. 15th International Conference on Computer-Aided Verification conference (CAV'O3)*, volume 2725 of *LNCS*, pages 27–39, 2003.

[B2] C. Eisner, D. Fisman, J. Havlicek, A. McIsaac, and D. Van Campenhout. The definition of a temporal clock operator. In *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 857–870. Springer, 2003.

[B3] C. Eisner, D. Fisman, J. Havlicek, and J. Martensson. The $\top, \bot$ approach to truncated semantics. Technical Report 2006.01, Accellera, 2006.

[B4] C. Eisner, D. Fisman, and J. Havlicek. A topological characterization of weakness. In *PODC '05: Proceedings of the Twenty-fourth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–8, New York, NY, USA, 2005. ACM.

[B5] D. Fisman. On the characterization of until as a fixed point under clocked semantics. In Proc. Third International *Haifa Verification Conference*, volume 4899 of *Lecture Notes in Computer Science*, pages 19–33. Springer, 2007.

[B6] C. Eisner and D. Fisman. Proposal for extending annex B of PSL with local variables, procedural blocks, past expressions and clock alignment operators. Technical Report H-0256, IBM, 2008.

[B7] C. Eisner and D. Fisman. Structural Contradictions. *Submitted*.

[B8] C. Eisner and D. Fisman. Augmenting a Regular Expression-Based Temporal Logic with Local Variables. FMCAD'08: Proc. 8th International Conference on Formal Methods in Computer-Aided Design, 2008.

[B10] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[B11] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.